

Parsing HTML in Java



WEB LANGUAGES - LAB

MATTEO TURRINI
(TURRINI@DISI.UNITN.IT)

A.Y.
2009 / 2010

Agenda



- Why do we need to build an HTML parser?
- HTML vs XML
- Parsing algorithms and libraries
- Overview of the API
- Implementation example
- Try to build a small parser
- Extension of the work, also with respect to the final project

Motivations behind HTML parser



- Let's imagine a common scenario: we want to retrieve information from the web using our application
- Ideal solution: an API interface is available, which, given a certain input, provide us with formatted data
- Common (bad) scenario: we have to crawl pages and, by parsing HTML, extract the information needed
- The latter is a poor solution because:
 - If the structure of the page changes, we have to change our parser too
 - No clear structure of the data, we have to figure out that
 - No RPC, we have to embed code directly

HTML vs XML



HTML

- Presentation purpose
- No semantic correlation between tags' names and content, tags define presentation (or navigation) features
- Maybe DTD only (DOCTYPE)
- No customization of tags
- Only some well-formed rules
- -

XML

- Custom data definition and exchange between applications
- Tags give us hints about the content
- External definition of entities, schema (.xsd) or dtd for validation
- Application-specific tags
- Well-formed rules
- Extensions: XQuery, XPath, XSLT, ...

Parsing algorithms and libraries



- To parse XML we have two popular and widely supported approaches: SAX and DOM (supported also in SDK 6 libraries)
- In HTML we have to use specific libraries
- There are different libraries available that can help us parsing HTML: we will present HtmlParser 1.6
- <http://htmlparser.sourceforge.net/>
- Similar approach used in XML DOM parsing

Steps for parsing an HTML page



1. Look at the HTML source of the page
2. Figure out the structure of the page (i.e. tags and attributes)
3. Find where the information we need have been put (which tags enclose them)
4. Get the page if not locally available
5. Extract the information using a parser

HtmlParser library



- Import the library into our java classpath
- Core classes we will use: [Parser](#), NodeFilter, NodeList, NodeIterator, Node
- Basically, the idea is to provide the parser class an HTML page (either locally or remotely available through an `HTTPURLConnection`) and to parse it according to given filters
- The function `parse` will return us the `NodeList` that corresponds to the filter criteria provided

An example: Springer (1)



[Content Types](#) [Subject Collections](#)

Institutional Login

Welcome!

To use the personalized features of this site, please [log in](#) or [register](#).

If you have forgotten your username or password, we can [help](#).

My Menu

[Marked Items](#)
[Alerts](#)
[Order History](#)

Saved Items

[All](#)
[Favorites](#)

Book Chapter



GenTax: A Generic Methodology for Deriving OWL and RDF-S Ontologies from Hierarchical Classifications, Thesauri, and Inconsistent Taxonomies

Book Series	Lecture Notes in Computer Science
Publisher	Springer Berlin / Heidelberg
ISSN	0302-9743 (Print) 1611-3349 (Online)
Volume	Volume 4519/2007
Book	The Semantic Web: Research and Applications
DOI	10.1007/978-3-540-72667-8
Copyright	2007
ISBN	978-3-540-72666-1
DOI	10.1007/978-3-540-72667-8_11
Pages	129-144
Subject Collection	Computer Science
SpringerLink Date	Saturday, June 23, 2007

PDF (248.1 KB) Free Preview

Martin Hepp¹ and **Jos de Bruijn¹**

(1) Digital Enterprise Research Institute (DERI), University of Innsbruck,
Abstract

Hierarchical classifications, thesauri, and informal taxonomies are likely the most valuable input for creating, at reasonable cost, non-toy ontologies in many domains. They contain, readily available, a wealth of category definitions plus a hierarchy, and they reflect some degree of community consensus. However, their transformation into useful ontologies is not as straightforward as it appears. In this paper, we show that (1) it often depends on the context of usage whether an informal hierarchical categorization schema is a classification, a thesaurus, or a taxonomy, and (2) present a novel methodology for automatically deriving consistent RDF-S and OWL ontologies from such schemas. Finally, we (3) demonstrate the usefulness of this approach by transforming the two e-business categorization standards eCl@ss and

An example: Springer(2)



```
import org.htmlparser.Parser;
import org.htmlparser.filters.HasAttributeFilter;
import org.htmlparser.util.NodeList;

public class Springer_Parser {

    private String URL;

    public Springer_Parser(String URL){
        this.URL= URL;
    }

    public String getTitle(){
        try{
            Parser parser = new Parser(URL);
            NodeList body = parser.parse (new HasAttributeFilter("class","Heading1"));
            return body.asString().toLowerCase();
        }
        catch(Exception e){
            System.err.println("Exception occured in the Springer page parsing methods: aborting process...");
            e.printStackTrace();
            return null;
        }
    }
}
```

An example: Springer(3)



```
public class ParserMain{

private static final String PROXY_URL = "proxy.science.unitn.it";
private static final String PROXY_PORT = "3128";

public static final void setProxy(){
    System.getProperties().put( "proxySet", "true" );
    System.getProperties().put( "proxyHost", PROXY_URL);
    System.getProperties().put( "proxyPort", PROXY_PORT);
}

public static void main(String[] args){
    Springer_Parser sp = new
        Springer_Parser("http://www.springerlink.com/content/w02834235t287l3v/");
    ParserMain.setProxy();
    System.out.println("Here is the title : " + sp.getTitle());
}
}
```

Try to build a small parser



- Let's make another example using a different page:
 - Extract the authors' last names from this URL:
<http://portal.acm.org/citation.cfm?id=343393.343400>

The screenshot shows the ACM Portal website interface. At the top left is the ACM logo and the word 'PORTAL'. To the right are links for 'Subscribe (Full Service)', 'Register (Limited Service, Free)', and 'Login'. Below this is a search bar with a dropdown menu showing 'The ACM Digital Library' (selected) and 'The Guide'. A 'SEARCH' button is to the right of the search bar. Below the search bar is a blue banner that reads 'THE GUIDE TO COMPUTING LITERATURE'. To the right of this banner is a 'Feedback' link with an icon. The main content area displays the title 'Verification of workflow task structures: A petri-net-based approach'. Below the title are fields for 'Source', 'Authors', 'Publisher', and 'Bibliometrics'. The 'Source' field includes 'Information Systems archive', 'Volume 25, Issue 1 (March 2000)', 'Pages: 43 - 69', 'Year of Publication: 2000', and 'ISSN:0306-4379'. The 'Authors' field lists 'Wil M.P. Van Der Aalst' and 'Arthur H. M. Ter Hofstede'. The 'Publisher' field is 'Elsevier Science Ltd. Oxford, UK, UK'. The 'Bibliometrics' field shows 'Downloads (6 Weeks): n/a', 'Downloads (12 Months): n/a', and 'Citation Count: 26'. Below these fields are links for 'Additional Information' (cited by, index terms, collaborative colleagues) and 'Tools and Actions' (Review this Article, Save this Article to a Binder, Display Formats: BibTex, EndNote, ACM Ref). At the bottom, there is a 'DOI Bookmark' field with the value '10.1016/S0306-4379(00)00008-9'. A section titled 'CITED BY 27' follows, listing three citations: 'Remco M. Dijkman, Marlon Dumas, Chun Ouyang, Semantics and analysis of business process models in BPMN, Information and Software Technology, v.50 n.12, p.1281-1294, November, 2008'; 'Jangha Cho, Cheng Hsu, A tool for minimizing update errors for workflow applications: the CARD model, Computers and Industrial Engineering, v.49 n.2, p.199-220, September 2005'; and 'Hyerim Bae, Wonchang Hu, Woo Sik Yoo, Byeong Kwon Kwak, Yeongho Kim, Yong-Tae Park, Document configuration control processes captured in a workflow, Computers in Industry, v.53 n.2, p.117-131, February 2004'.

Solution



```
import java.util.ArrayList;

import org.htmlparser.Parser;
import org.htmlparser.filters.HasAttributeFilter;
import org.htmlparser.filters.TagNameFilter;
import org.htmlparser.util.NodeList;
import org.htmlparser.util.SimpleNodeIterator;

public class ACM_Parser {

    public ArrayList<String> getAuthorsLastName(){
        ArrayList<String> authorsList = new ArrayList<String>();
        String lastName = "-";
        try{
            Parser parser = new Parser(URL);
            NodeList body = parser.parse (new HasAttributeFilter("class","authors"));
            NodeList authors = body.extractAllNodesThatMatch(new TagNameFilter("a"), true);
            SimpleNodeIterator iter = authors.elements();
            while(iter.hasMoreNodes()){
                lastName = iter.nextNode().getFirstChild().getText();
                lastName = lastName.substring(lastName.lastIndexOf(" ") + 1);
                authorsList.add(lastName);
            }
            return authorsList;
        }
        catch (Exception e){
            System.err.println("Exception occurred in the ACM page parsing methods: aborting process...");
            e.printStackTrace();
            return null;
        }
    }
}
```

Publish And Perish Evaluation Tool



**DETAILS ABOUT IT AND POTENTIAL
EXTENSIONS**

What is this tool about?



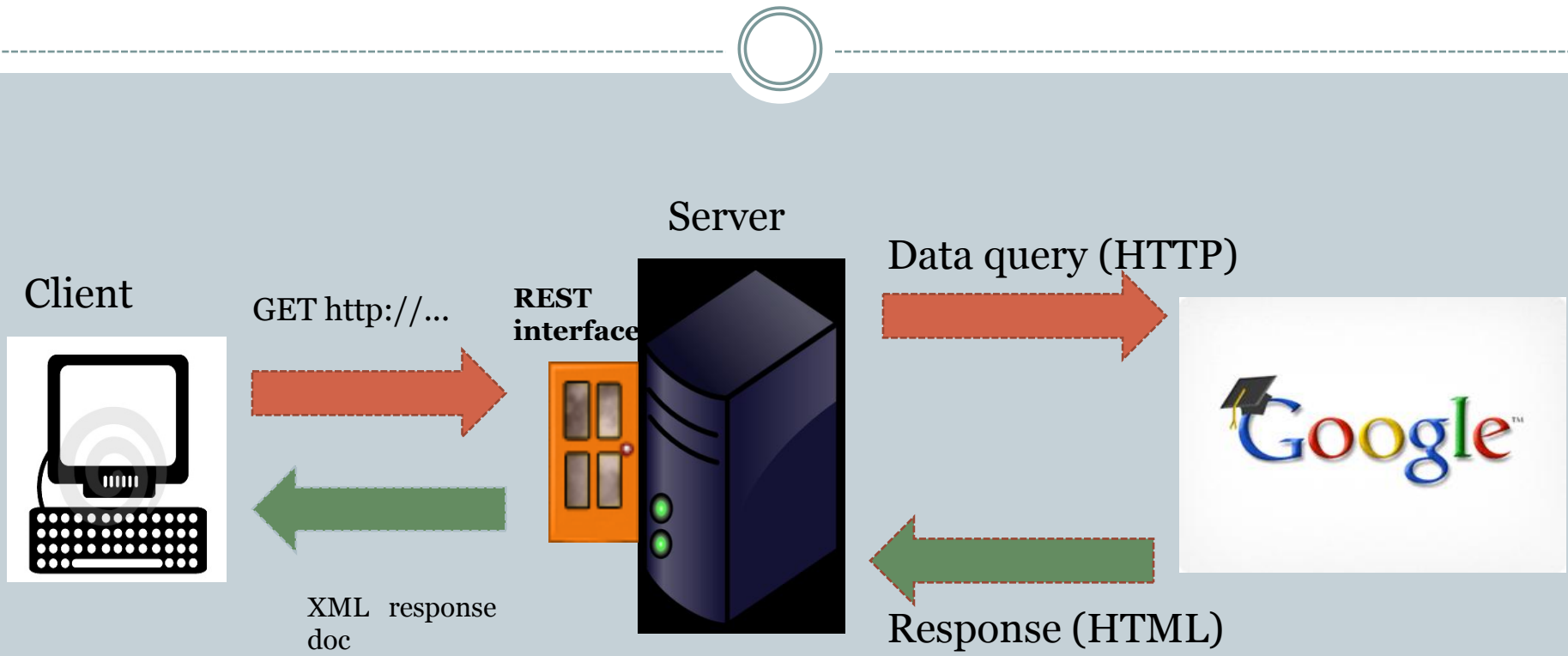
- With this tool we want to retrieve information about academic and research people from Google Scholar
- These information are:
 - List of publications made and details about each of them
 - List of co-authors
 - Citations received per publication
 - Years of activity
 - Publishers who published the articles and papers written by the specified author
 -
- Since there are no API for GS, we have to parse directly the HTML of GS response page about the requested author
- Starting from the data retrieved we make some further computations and statistics

Why do we need a REST interface?



- The goals of having a REST interface associated to our tool are:
 - Have a set of procedures remotely callable
 - Get the information produced as output by the tool in a structured format (in this case XML)
 - In general, give the user the possibility to use our service without having to deal with any details about how it is structured or implemented
 - (Maybe) reuse it in order to build more complex services

Sample scenario



Request details (1)



- In order to get the information for a researcher (remember, CS and ENG only) we have to insert this URL:

http://wiki.liquidpub.org:8080/Evaluation_Tool/resources/BasicStats/{firstName}/{middleName}/{lastName}

where ***firstName***, ***middleName*** and ***lastName*** are related to the researcher we want to get information about

NB!

Remember that in HTTP URLs, if you want to insert a blank space (for instance if the researcher doesn't have a middleName), you have to type the special string code "%20"

Response details (1)



```
<stats>
  <metric>
    <name>h-index</name>          -> Hirsch index (citation based)
    <value>3</value>
  </metric>
  <metric>
    <name>g-index</name>         -> A modified Hirsch index (citation based)
    <value>5</value>
  </metric>
  <metric>
    <name>signal-to-noise</name> -> ratio between nr° of publications that contributed to the calculation of
    the H-index and overall nr° of publications
    <value>0.664</value>
  </metric>
  <metric>
    <name>citationsCount</name>
    <value>51</value>
  </metric>
  <metric>
    <name>numberOfPublications</name>
    <value>21</value>
  </metric>
  <metric>
    <name>avgCitations</name>
    <value>5.34</value>
  </metric>
</stats>
```

Response details (2)



```
<listOfPublications>
  <item>
    <title>Interesting paper about web services and SOA</title>
    <authorsList>
      <author>
        <firstName>Fabio</firstName>
        <middleName>-</middleName>
        <lastName>Casati</lastName>
      </author>
    </authorsList>
    <year>2001</year>
    <citations>15</citations>
  </item>
</listOfPublications>

</stats>
```

Parsing the XML



- To extract the information from the XML document we can use different XML parsers and libraries
- SAX -> usually more efficient (sequential, event-based), but also more difficult to code
- DOM -> less efficient (the whole tree is stored in the main memory), but easier to code + document navigation is more intuitive
- Suggested library: standard Java 6

Warning about the usage



- Remember that the results are computed by sending queries directly to Google Scholar
- We cannot send from a single IP address too many requests, because in that case our queries will be considered as a DoS attack
- Therefore, Google will “ban” our requesting host for a limited amount of time, rejecting all the requests coming from it